PowerShell and the Import-Csv CmdLet – A New PowerShell Series

Managing data inside a CSV file is surprisingly simple thanks to the PowerShell **Import-CSV** CmdLet. I'm often presented with data from different sources in a variety of formats, and by and large the common format used for data extracts is CSV. Regardless of the system that generates the content, we can all render and consume data using this simple and effective format.

You will find the Microsoft TechNet documentation here:
http://technet.microsoft.com/en-us/library/dd347665.aspx

PowerShell Import-CSV into Variable

When it comes to using PowerShell Import-CSV the best way to illustrated it is to use an example. In this example, we will use PowerShell Import-CSV to assign our data to a variable of the array type.

The file that we will use as our example is one that contains a username, email address, street address, postal code (zip code for my US friends) and a phone number. This is what the file would look like for our sample which will be named **UserList.CSV**:

> *username,full name,email,address,postalcode,phone*
> *ewright,Eric Wright,eric@somedomain.com,123 Any Street,L4C 1N8,555-1212*
> *jvoigt,Jens Voigt,jens@somedomain.com,456 Any Street,L4C 1N8,555-3456*
> *ppocklington,Peter Pocklington,peter@anotherdomain.com,890 Back Street,M2H 4Y1,555-9876*

Note that there is a header row which is one of the most important aspects for our script. If the file that you want to use does not have a header row, we can add one. I'll go into that more deeply in an upcoming article.

What we do first is to bring the file using PowerShell Import-CSV into a variable as an array. This is done by creating a variable which we will call **$userobjects** and assigning it a value using the **Import-CSV** CmdLet. Let's assume that you have the import file in a folder **X:ImportData**

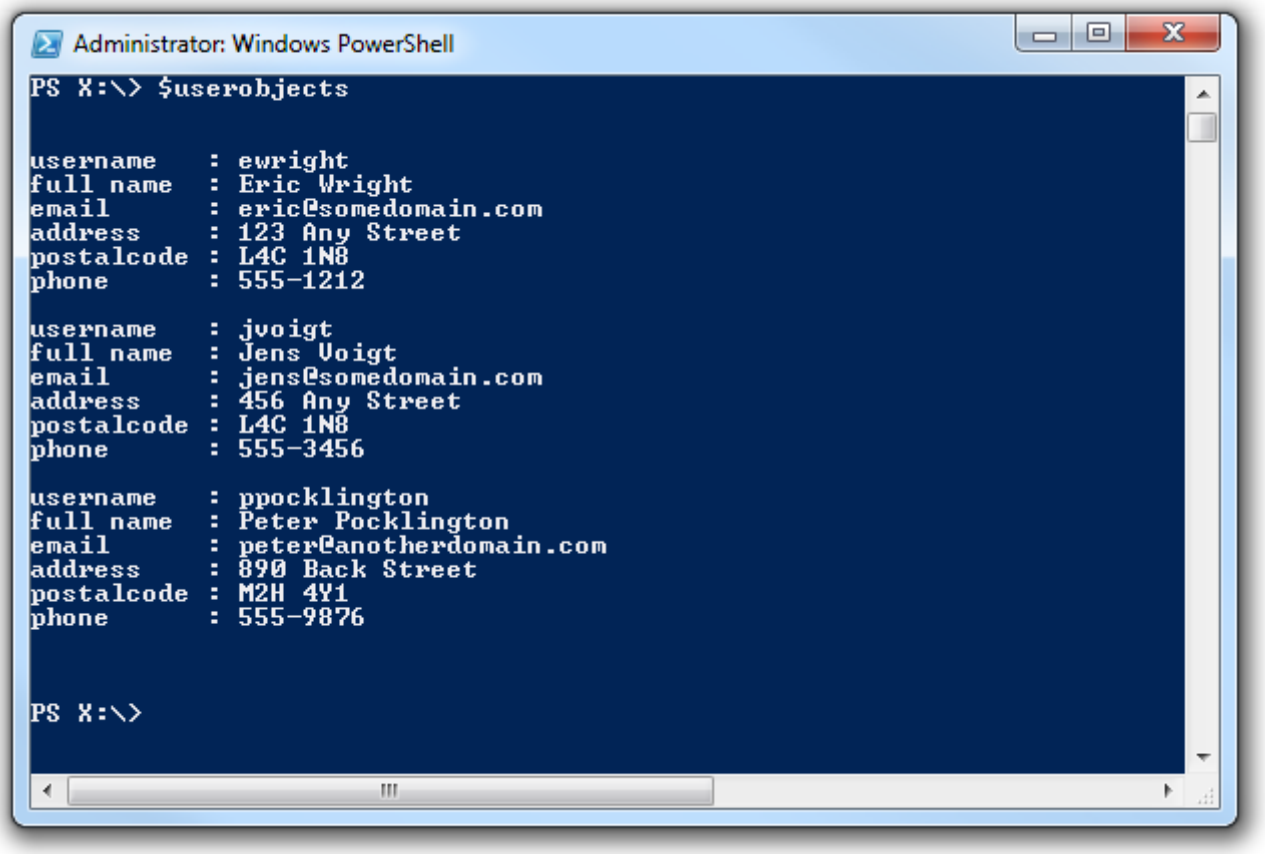
```
$userobjects = Import-CSV x:ImportDataUserList.CSV
```

No really, it's just that simple. Now you have an array variable which you can use to access the data in a number of ways. Let's look at a couple of methods to pick out data:

Accessing Data After Importing CSV into an Array Varriable

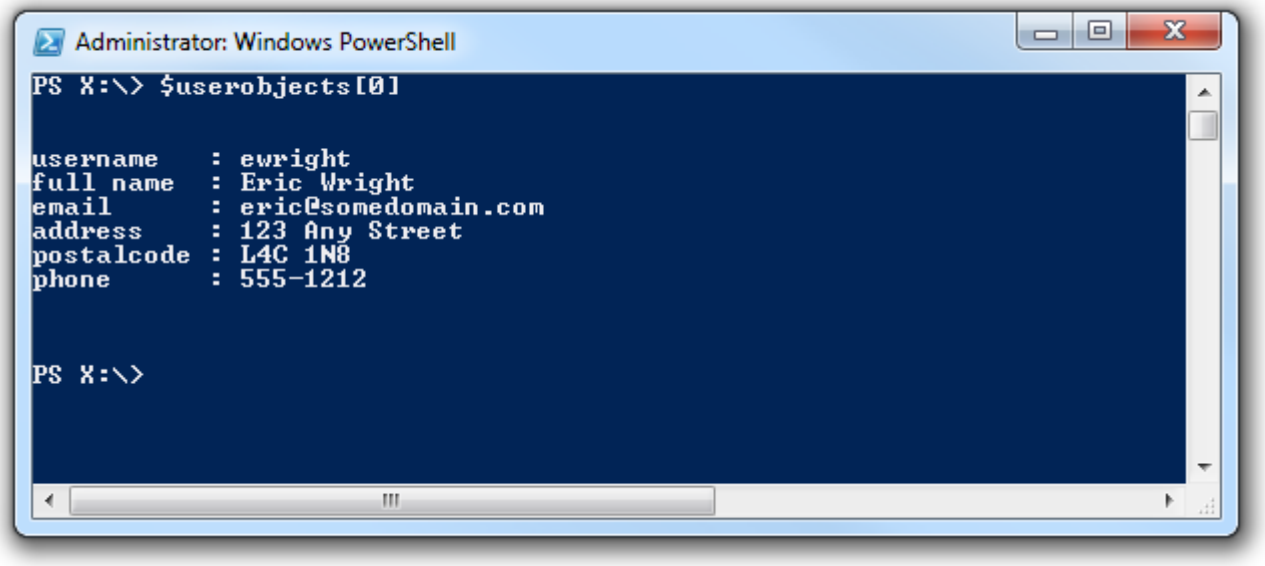
Display all instances in the array:

```
$userobjects
```



Display the first element in the array:

```
$userobjects[0]
```

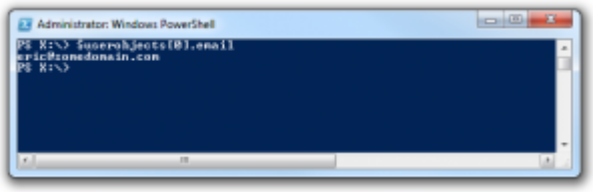


Ok, hold on a second. Why did we use 0 (zero) to pull the first record? This is a done because when an array is populated, the first element is in position 0. For a 2 dimensional array, the first element is 0,0 rather than 1,1 which can be a source of confusion.

The purpose of the article was to talk about headers so now you will see where that comes into play. We have drawn out the different elements inside the array, so now we can use the headers to be able

to gather the content of each element in a meaningful way. Let's say that we want to see the email address of the first element.

```
$userobjects[0].email
```



It's so dangerously simple really. Because the array was imported with the Import-CSV, the header row was assigned to columns just as we would see it inside a spreadsheet program. Just think of it as **element[instance].property** which means you can select any assigned property (column header) from any instance of the element.

We can also do things such as count the elements which can be a handy piece of information to have:

```
$userobjects.count
```

In part 2 we will expand our script to loop through the elements and show you different ways that we can manage our data from the CSV file and begin to perform other operations using the contents.

This post is part of a series on using the PowerShell Import-CSV Command.

Part 1 – Importing a CSV Into PowerShell with Import-CSV

Part 2 – Looping when Importing a CSV Into PowerShell with Import-CSV foreach

Part 3 – Importing a CSV Into PowerShell with Import-CSV and Adding Headers